REMARKS

In the July 2, 2003, third Office Action in this application, the United States Patent and Trademark Office rejected Claims 4, 9, 13, and 14-19 under 35 U.S.C. § 112, second paragraph, as being indefinite. Claims 1, 3-5, 7-9, and 11-14 were rejected under 35 U.S.C. § 102(e) as being anticipated in view of the teachings of U.S. Patent No. 5,619,710 issued to Travis, Jr., et al. (hereinafter "Travis et al."). Claims 2, 6, 10, and 15 were rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Travis et al. taken in view of the teachings of U.S. Patent No. 5,734,903 issued to Saulpaugh et al. Claims 18 and 19 were rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Travis et al. This amendment amends the term "medium" recited in Claims 9, 13, and 15-19 to include "computer-readable medium" so as to clarify the claimed invention, thereby rendering moot the rejection of Claims 9, 13, and 15-19 under 35 U.S.C. § 112, second paragraph.

Prior to discussing in detail why applicant believes that all of the claims in this application are allowable, a brief description of applicant's invention and a brief description of the teachings of the cited and applied references are provided. The following background and the discussions of the disclosed embodiments of applicant's invention and the teachings in the cited and applied references are not provided to define the scope or interpretation of any of the claims of this application. Instead, such discussions are provided to help the Office better appreciate important claim distinctions discussed thereafter.

Summary of Processes and Threads

Early operating systems allowed users to run only one program at a time. Users ran a program, waited for it to finish, and then ran another one. Modern operating systems allow users to execute (run) more than one program at a time or even multiple copies of the same program at the same time. This change highlights a subtle distinction: the difference between a program

-6-

and a process. A program is a static sequence of instructions whereas a process is the dynamic invocation of a program along with the system resources required for the program to run. Thus, a process, in the simplest terms, is an executing program.

Processes include one or more threads. A thread is the basic unit used by the operating system to allocate processor time. A thread can include any part of the process code, including parts currently being executed by another thread. A processor is capable of executing only one thread at a time. However, a multitasking operating system, i.e., an operating system that allows users to run multiple programs, appears to execute multiple programs at the same time. In reality, a multitasking operating system continually alternates between programs, executing a thread from one program, then a thread from another program, etc. As each thread finishes its subtask, the processor is given another thread to execute. The extraordinary speed of the processor provides the illusion that all of the threads execute at the same time. Multitasking increases the amount of work a system accomplishes because most programs do not require that threads continuously execute. For example, periodically, a thread stops executing and waits while a slow resource completes a data transfer or while another thread is using a resource it needs. When one thread must wait, multitasking allows another thread to execute, thus taking advantage of processor cycles that would otherwise be wasted.

While the terms multitasking and multiprocessing are sometimes used interchangeably, they have different meanings. Multiprocessing requires multiple processors. If a machine has only one processor, the operating system can multitask, but not multiprocess. If a machine has multiple processors, the operating system can both multitask and multiprocess.

In summary, an operating system having a process to run presumes the existence of a program from which the process is invoked because the program is the static aspect of the process and the process is the dynamic aspect of the program. Programs are typically created in

MSFT\17210AM3.DOC

an application development environment using an application development language. An application development environment is an integrated suite of applications for use by software developers to create programs. Typical components of application development environments include a compiler, a file browsing system, a debugger, and a text editor for use in creating programs. An application development language is a computer language designed for creating programs. The present invention as summarized below focuses on an application development environment, which in other words is a programming environment.

Summary of the Invention

Applicant's invention is directed to improving the development of programs so that programs may be developed in a more efficient and bug-free manner. As programs have become increasingly complex, programming environments are becoming a factor in the length of time it takes to create new programs and the number of bugs that occur as new programs are executed in conjunction with other programs. Traditionally, programs are developed in a linear manner. Unfortunately, a linear programming environment is often undesirable when highly complex computer programs that must run concurrently with other programs are to be developed. Another programming environment is the message-driven environment. In a message-driven programming environment, different objects interface with other objects via messages. Such messages typically are complex structures. To use such messages, the context must be unpacked from a message prior to the execution of an action. The message-driven programming environment as well as the linear programming environment are fragile and render development of programs that must run concurrently with other programs potentially more difficult and bug-laden than is desired.

To solve or to reduce the foregoing problems, applicant's invention provides an asynchronous programming environment. More specifically, applicant's invention provides a

-8-

dynamic object storage scheme for storing a plurality of objects, a dynamic dispatch scheme based on the plurality of objects, and an object recognition scheme to describe the plurality of objects. The dynamic object storage scheme stores the plurality of objects so that each object can be accessed as necessary by different threads within the asynchronous programming environment. The dynamic object storage scheme is dynamic in that objects can be created and removed as necessary during the execution of tasks within the asynchronous programming environment. The dynamic dispatch scheme is designed to invoke an action based on the plurality of objects stored by the dynamic object storage scheme. Each action that is invoked by the dynamic dispatch scheme falls into one of a plurality of categories. Actions may be classified differently at different times.

The plurality of categories include three categories—actions needing precisely one object, actions needing more than one object, or actions not needing an object at all. Actions needing precisely one object include message handlers, and other actions that do not need any object for their execution are generally used to create objects, such as default constructors and real-time routines. Actions that require multiple objects for their dispatch generally combine objects and perform tasks as designed by the programmer.

Other aspects of the present invention include an object recognition scheme that describes the plurality of objects stored by the dynamic object storage scheme. The description of objects allows the asynchronous programming environment to determine whether a described object fits the needs of an application programming interface. In other words, when an application programming interface is presented to the asynchronous programming environment, it is not always clear which objects in a depository of objects fit the presented application programming interface. The object recognition scheme of the present invention allows one or more objects to

-9-

be identified, thereby determining the suitability of one or more objects to the requirements of the application programming interface.

Unlike prior programming environments, an asynchronous programming environment allows thread-agnostic programs to be developed. Such programs have symmetric multithreading capabilities, i.e., they can be scaled to use the most of available processor processing power. Symmetric multithreading is expected to reduce instruction cache misses on hardware implementations that handle multiple instruction streams at once. An asynchronous programming environment is more efficient than a message-driven programming environment because execution is driven via presence of objects so that little or no translation is needed to recover context of execution. These attributes of an asynchronous programming environment allow programs to be developed in a more efficient and more bug-free manner.

Summary of Travis et al.

Travis et al. is directed to a system for invoking an application across a network. The network of Travis et al. contains several platforms connected by the network. Each platform includes a central processing unit and memory. Running on each central processing unit of each platform is an operating system, applications, and an application control architecture services (ACAS) software component. The ACAS software components of the various platforms of Travis et al., via an object-oriented technique, allow one application on one platform to invoke another application on another platform. The memory of each platform contains information for a global database which is shared by all platforms over the network.

The system of Travis et al. organizes applications, data of applications, and operations on data of applications into object-oriented classes. As discussed by Travis et al. at Figure 2, an application, such as a word processor, can be conceptualized into two parts: application definitions and application data. If the application is a word processing application, the

MSFT\17210AM3.DOC

application definitions include operations that the word processing application can perform and the type of data on which the word processing application can operate. The application data includes specific data (and not the type of data) on which the word processing application can operate. The application definitions and the application data are characterized in terms of object types, and are referred to by Travis et al. as objects. The application definitions can be organized into class objects and method objects. Class objects are names, which can be used to describe the application. The method objects describe types of operations that are performed by the application. Instance objects are derived from the application data, which can be manipulated or accessed by the class objects and the method objects.

An example of an instance object is a specific file, such as a file called MYFILE. The file MYFILE is operated on by a specific WRITE application. Because MYFILE is a specific file, it is not handled by the ACAS software components. MYFILE belongs to a class of compatible files, such as ASCII_FILE, which is handled by the ACAS software components. Similarly, the specific WRITE application is not handled by the ACAS software components. Instead, the specific WRITE application belongs to a class of applications called ALLWRITE, which in turn is handled by the ACAS software components. Applications can be characterized by the class objects to which the applications belong, by the method objects which support the operations in a particular application, and by instance objects upon which the method objects can operate.

Each instance object is associated with a class object. An instance object is manipulated by one class object (a first application) sending a message to another class object (a second application). The message might be an action, such as EDIT, READ, or PRINT. Messages are supported by the class objects. This means that the interpretation of a message depends upon the classes to which the instance objects in that message belong. For example, a PRINT message

may be interpreted differently if the instance object is a text file in a class object TEXT_FILES as opposed to an instance object that is a color graphics file in a class object COLOR_GRAPHICS. Thus, messages are the interfaces between a class object (an application) and method objects (operations of the application). Messages are used against an application to specify types of operations the application can perform on the instance objects. The messages are generally in the form of a selector of an operation, such as PRINT, along with several parameters (e.g., STRINGS, NUMBERS, etc.). A message does not describe the implementation of a particular operation. It only represents the interface to the implementation of the particular operation. Thus, to find a particular operation (the method object) that is called by a particular message, one must not only examine the message but also the class of the instance object. To cause a specific action to occur, the message must be mapped to actual executable program code. This mapping occurs by finding the particular message which corresponds to the particular class of the particular instance and then finding the particular method which corresponds to the message supported by the class. The method represents the actual executable program code to implement the desired operation of the message on the instance.

The essence of the system of Travis et al. is to have a mechanism for a message to be sent from one class object (an application) on one platform to invoke an action (a method object) in another class object (another application) on a different platform.

Summary of Saulpaugh et al.

Saulpaugh et al. is directed to a system for object-oriented message filtering and for selectively transferring a message between a client task and one or more server tasks for preprocessing, processing, and post-processing. The system of Saulpaugh et al. comprises an object database having a filter object memory, an object management unit, a message transaction unit, and a locking unit. The object management unit creates a port object in one or more

MSFT\17210AM3.DOC

associated target message objects. The object management unit selectively creates one or more filter objects associated with a target message object, and selectively associates a pre-processor message object, a post-processor message object, or both a pre-processor message object and a post-processor message object with each filter object. The message transaction unit selectively routes a message sent by a client task and directed to a target message object to one or more associated pre-processor message objects prior to delivering a message to the target message object.

After delivery to the target message object, the message transaction unit selectively routes the message to one or more of the associated post-processor message objects. Server tasks receive messages from port objects associated with pre-processor message objects, target message objects, and post-processor message objects. The server tasks facilitate pre-processing, processing, and post-processing operations. Message object locking and unlocking operations are performed by the locking unit.

The Claims Distinguished

As discussed in greater detail below, the claims of the present application are clearly patentably distinguishable over the teachings of the above-cited references. The present invention is directed to provide an asynchronous programming environment, which solves or reduces problems associated with prior programming environments. Applicant's invention provides a dynamic object storage scheme for storing a set of objects, a dynamic dispatch scheme based on the set of objects, and an object recognition scheme to describe the set of objects. The dynamic object storage scheme stores the set of objects so that each object can be accessed as necessary by different threads within the asynchronous programming environment. The dynamic object storage scheme is dynamic in that objects can be created and removed as necessary during the execution of tasks within the asynchronous programming environment.

The dynamic dispatch scheme is designed to invoke an action based on the plurality of objects stored by the dynamic objects storage scheme. The action that is invoked by the dynamic dispatch scheme falls into one of a plurality of categories. Action may be classified differently at different times. The plurality of categories of actions include three categories--actions needing precisely one object, actions needing more than one object, or actions not needing any object at all. Actions needing precisely one object include message handlers, and other actions that do not need any objects for the execution are generally used to create objects, such as default constructors and real-time routines. Actions that require multiple objects for the dispatch generally combine objects and perform tasks as designed by the developer. The object recognition scheme describes the set of objects stored by the dynamic objects storage scheme. The description of objects allows the asynchronous programming environment to determine whether a described object fits the needs of an application programming interface. In other words, when an application programming interface is presented to the asynchronous programming environment, it is not always clear which objects in a depository of objects fit the presented application programming interface. The object recognition scheme of the present invention allows one or more objects to be identified, thereby determining the suitability of one or more objects to the requirements of the application programming interface.

Regarding the claims, independent Claim 1 is directed to an asynchronous programming environment. The environment is recited as comprising a dynamic object storage scheme for storing a plurality of objects. The environment further comprises a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no object. The environment yet further comprises an object recognition scheme for providing a description of

-14-

each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 2-4 are dependent from independent Claim 1 and are directed to further limitations of the environment described above. Claim 2 is dependent on Claim 1 and recites that in the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 3 is dependent on Claim 1 and recites that the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 4 is dependent on Claim 1 and recites that the dynamic dispatch scheme provides for execution of objects based on unpacked-into-messages events.

Independent Claim 5 is directed to a method, and the method is recited as comprising storing a plurality of objects via a dynamic object storage scheme. The method further comprises dispatching at least one of the plurality of objects via a dynamic dispatch scheme based on events from at least one of the plurality of objects. The dynamic dispatch scheme is capable of invoking an action that belongs to one of a plurality of categories. The pluralities of categories include needing one object, needing more than one object, and needing no object. The method yet further comprises describing each of the plurality of objects utilizing an object recognition scheme. The object recognition scheme provides a description of each object of the plurality of objects. The description allows a determination of whether the object described by the description fits an application programming interface. Claims 6-8 are dependent from independent Claim 5 and are directed to further limitations of the method described above. Claim 6 is dependent on Claim 5 and recites that the act of storing a plurality of objects via a dynamic objects storage scheme comprises accessing one of the plurality of objects utilizing a recyclable locking mechanism. Claim 7 is dependent on Claim 5 and recites that the act of

MSFT\17210AM3.DOC

describing each of the plurality of objects utilizing an object recognition scheme comprises describing each of the plurality of objects as a series of tokens. Each token relates to an attribute of the object. Claim 8 is dependent on Claim 5 and recites that the act of dispatching at least one of the plurality of objects via a dynamic dispatch scheme comprises executing at least one of the plurality of objects based on unpacked-into-messages events.

Independent Claim 9 is directed to a computer, and the computer is recited as comprising a processor; a computer-readable medium; and an asynchronous programming environment being executed by the processor from the computer-readable medium. The environment is recited by independent Claim 9 as comprising a dynamic object storage scheme for storing a plurality of objects. The environment is further recited as comprising a dynamic dispatch scheme based on an event from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing on object, needing more than one object, and needing no object. The environment yet further comprises an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 10-13 are dependent from independent Claim 9 and are directed to further limitations of the computer described above. Claim 10 is dependent on Claim 9 and recites that the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 11 is dependent on Claim 9 and recites that in the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 12 is dependent on Claim 9 and recites that the dynamic dispatch scheme provides for the execution of objects based

MSFT\17210AM3.DOC

on unpacked-into-messages events. Claim 13 is dependent on Claim 9 and recites that the medium comprises a piece of memory.

Independent Claim 14 is directed to a computer-readable medium that has a computer program stored thereon for execution on a computer. The computer program is recited by independent Claim 14 as providing an asynchronous programming environment. The environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment further recites a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object. The environment yet further recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 15-19 are dependent from independent Claim 14 and are directed to further limitations of the computer-readable medium described above. Claim 15 is dependent on Claim 14 and recites that the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 16 is dependent on Claim 14 and recites that in the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 17 is dependent on Claim 14 and recites that the dynamic dispatch scheme provides for execution of objects based on unpacked-into-messages events. Claim 18 is dependent on Claim 14 and recites that the computer-readable medium comprises a Compact Disc Read-Only Memory (CD-ROM). Claim 19 is dependent on Claim 14 and recites that the computer-readable medium comprises a floppy disk.

MSFT\17210AM3.DOC

As noted above, the third Office Action rejected Claims 1, 3-5, 7-9, and 11-14 under 35 U.S.C. § 102(e) as being anticipated in view of the teachings of Travis et al. Claims 2, 6, 10, and 15 were rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Travis et al., taken in view of the teachings of Saulpaugh et al. Additionally, Claims 18 and 19 were rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Travis et al. As also noted above, applicant respectfully disagrees. The cited and applied references simply fail to teach all of the limitations of the independent claims, much less the recitations of many of the dependent claims.

Focusing on Claim 1, there is no teaching or suggestion in the cited references for an asynchronous programming environment in the manner recited in Claim 1. Claim 1 recites that the environment comprises a dynamic object storage scheme for storing a plurality of objects. The remaining recitations of Claim 1 are directed to a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories. The plurality of categories are recited by Claim 1 to include needing one object, needing more than one object, and needing no object. Claim 1 also recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. The cited and applied references have failed to teach the concept of a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories (which includes needing one object, needing more than one object, and needing no object), among other things.

Travis et al. describes a mechanism where client applications can remotely invoke other applications by sending globally recognized (network wide) messages with parameters. Using the name of the message and information about certain parameters and certain preference information, one method object is selected from a database. Other information in the database

MSFT\17210AM3.DOC

can be used to locate and execute the actual code to implement the method object. <u>The action represented by the message is resolved into one method object and executes as specified in the message</u>.

In contrast, applicant's invention recites, among other things, a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, <u>needing more than one object,</u> and needing no object. Whereas the system of Travis et al. invokes one method object in response to a message, the dynamic dispatch scheme of various embodiments of the present invention allows the invocation of an action that belongs to one of a plurality of categories, including needing one object, needing more than one object, and needing no object. Among other things, the system of Travis et al. has failed to invoke an action that needs more than one object.

Travis et al. specifically states that in response to a message requesting a method invocation from an application or user, a client application determines the proper method to be invoked by retrieving information from a class database, comparing the retrieved information with user preferences, and selecting the proper method based upon the comparison. Server connection and startup processes locate a platform capable of executing code associated with the selected method. <u>In other words, the system of Travis et al. is not designed to invoke an action that requires multiple objects</u>. Because Travis et al. has failed to disclose all claimed limitations or features of the claimed invention, no *prima facie* case of anticipation has been established.

The Office cited Saulpaugh et al. for the teaching of the claimed "recyclable locking mechanism." This cannot be correct. The Office explained that the recyclable locking mechanism is disclosed by locking unit 46 of the system of Saulpaugh et al. In the preferred embodiment of Saulpaugh et al., each element in a list of locked message objects is a lock structure that specifies a message object ID and a semaphore. The locking unit 46 of Saulpaugh

-19-

et al. locks a message object by inserting a new lock structure containing the corresponding message object ID into the list of locked message objects. To remove a message object that is currently locked, the locking unit 46 of Saulpaugh et al. removes the corresponding lock structure from the corresponding port object's list of locked message objects, thereby unlocking the message object. In other words, the system of Saulpaugh et al. creates a new lock structure for each message object to be locked; hence, the locking structure of Saulpaugh et al. is not recyclable.

What makes applicant's recyclable locking mechanism recyclable is that there exists a pool of a limited number of locks. When the lock is no longer necessary, the lock returns to the pool so that it can be reused, and hence is recyclable. In the case of Saulpaugh et al., in order for a message object to be locked, a new lock structure is created. To put it simply, each message object in Saulpaugh et al. must have its own lock structure in order to participate in synchronization. The problem with Saulpaugh et al. is that the overhead of a lock in each message object would draw precious computing resources in order to provide a maintenance of the lock. No lock is recyclable in the system of Saulpaugh et al. because each message object needs its own lock. Not so with applicant's invention. Given the defects of Saulpaugh et al., there would be no benefit to combine Travis et al. and Saulpaugh et al. Even if the combination were possible, which assumption applicant specifically denies, the combination still would not teach applicant's invention.

Clearly, neither Travis et al. nor Saulpaugh et al., alone much less in combination, teaches or suggests the subject matter of Claim 1. More specifically, none of these references, alone much in combination, teaches or suggests a asynchronous programming environment that has a dynamic dispatch scheme, which invokes an action that belongs to one of a plurality of categories (needing one object, needing more than one object, and needing no object), and an

object recognition scheme, which provides a description of each object of the plurality of objects so as to allow a determination of whether an object described by the description fits an application programming interface, as recited in Claim 1.

As will be readily appreciated in the foregoing discussion, none of the cited and applied references teaches or suggests the subject matter of Claim 1. Specifically, none of the cited and applied references teaches an object recognition scheme in the manner recited in Claim 1. As a result, applicant submits that Claim 1 is clearly allowable in view of the teachings of the references.

With respect to dependent Claims 2-4, all of which depend directly or indirectly from Claim 1, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely Travis et al. or Saulpaugh et al. Claims 2-4 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with the recitations of the claims from which these claims individually depend. In summary, Claims 2-4 are submitted to be allowable for reasons in addition to the reasons why Claim 1 is submitted to be allowable.

Independent Claim 5 is directed to a method, which recites an act for storing a plurality of objects via a dynamic object storage scheme. The method further recites an act for dispatching at least one of the plurality of objects via a dynamic dispatch scheme based on events from at least one of the plurality of objects. The dynamic dispatch scheme is capable of invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no object. The method of Claim 5 yet further recites an act for describing each of the plurality of objects utilizing an object recognition scheme. The object recognition scheme provides a description of each object of the plurality of objects. The description allows a determination of whether an object described by

-21-

the description fits an application programming interface. For generally the same reasons discussed above with respect to Claim 1, applicant submits that the subject matter of Claim 5 is not taught or suggested by any of the cited and applied references, and thus, that Claim 5 is also allowable.

With respect to dependent Claims 6-8, all of which depend directly or indirectly from Claim 5, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely, Travis et al. or Saulpaugh et al. Claims 6-8 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with these recitations of the claims from which these claims individually depend. In summary, Claims 6-8 are submitted to be allowable for reasons in addition to the reason why Claim 5 is submitted to be allowable.

Independent Claim 9 is directed to a computer. The computer recites a processor, a computer-readable medium, and an asynchronous programming environment being executed by the processor from the medium. The computer of Claim 9 further recites that the environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment yet further recites a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no object. The environment additionally recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. For generally the same reasons discussed above with respect to Claims 1 or 5, applicant submits that the subject matter of Claim 9 is not taught or suggested by any of the cited and applied references, and thus, that Claim 9 is also allowable.

-22-

With respect to dependent Claims 10-13, all of which depend directly or indirectly from Claim 9, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely, Travis et al. or Saulpaugh et al. Claims 10-13 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with the recitations of the claims from which these claims individually depend. In summary, Claims 10-13 are submitted to be allowable for reasons in addition to the reasons why Claim 9 is submitted to be allowable.

Independent Claim 14 is directed to a computer-readable medium that has a computer program stored thereon for execution on a computer. The computer program is recited by Claim 14 as providing an asynchronous programming environment. This environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment is further recited to include a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no objects. The environment as recited by Claim 14 further includes an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. For generally the same reasons discussed above with respect to Claims 1, 5, or 9, applicant submits that the subject matter of Claim 14 is not taught or suggested by any of the cited and applied references, and thus, that Claim 14 is also allowable.

With respect to dependent Claims 15-19, all of which depend directly or indirectly from Claim 14, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely Travis et al. or Saulpaugh et al. Claims 15-19 all add limitations that are clearly not taught or suggested by any of the cited and applied references,

MSFT\17210AM3.DOC

particularly when the limitations are considered in combinations with the recitations of the claims from which these claims individually depend. In summary, Claims 15-19 are submitted to be allowable for reasons in addition to the reasons why Claim 14 is submitted to be allowable.

The Second Request to Correct Inventorship Under M.P.E.P. § 605.04(g)

Given that the previous request made by the applicant was completely ignored by the Office, the Office is again notified of a typographical or transliteration error in the spelling of the inventor's name. The incorrect inventor's name of "Slovak Ondrej Such" should read "Ondrej Such." In the originally-submitted declaration of December 21, 1998, the name of the inventor in the present patent application is set forth as "Ondrej Such." The inventor's citizenship was incorrectly set forth as "Czech Republic." The incorrect citizenship designation "Czech" was crossed out and the correct citizenship designation "Slovak" was handwritten so as to rectify the incorrect citizenship designation. The Office incorrectly interpreted, as evidenced by the filing receipt received on January 28, 1999, that the name of the inventor is "Slovak Ondrej Such." Applicant respectfully requests that this incorrect name designation be changed to the correct name designation of "Ondrej Such." More particularly, applicant respectfully requests that the Examiner have the Technology Center's technical support staff enter the correct name designation "Ondrej Such" in the PALM database or other pertinent databases and print out a new bibliographic data sheet, which should be placed in the file wrapper of the present patent application.

The Term "Unpacked-Into-Messages"

The Office has rejected Claim 4 under 35 U.S.C. § 112, second paragraph, because the term "unpacked-into-messages" is a relative term which allegedly renders Claim 4 indefinite. The Office further indicated that the term "unpacked-into-messages" is not defined by the claim or by the specification. Applicant respectfully disagrees. The term "unpacked-into-messages"

MSFT\17210AM3.DOC

qualifies an event to differentiate such an event from events that are packed into messages. See the pending patent application, page 16, lines 12-16, among other places. Because Claim 4 is sufficiently definitive, the rejection of Claim 4 should be withdrawn. Reconsideration and allowance of Claim 4 is respectfully requested.

CONCLUSION

In view of the foregoing remarks, applicant submits that all of the claims in the present application are clearly patentably distinguishable over the teachings of Travis et al. and Saulpaugh et al. Thus, applicant submits that this application is in condition for allowance. Reconsideration and re-examination of the application and allowance of the claims and passing of the application to issue at an early date are solicited. If the Examiner has any remaining questions concerning this application, the Examiner is invited to contact the applicant's undersigned attorney at the number below.

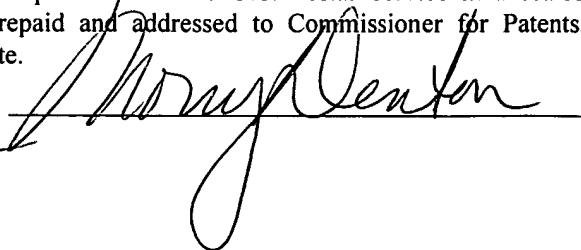Respectfully submitted,

CHRISTENSEN O'CONNOR
JOHNSON KINDNESS PLLC

D.C. Peter Chu
Registration No. 41,676
Direct Dial No. 206.695.1636

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid and addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the below date.

Date: October 31, 2003

DPC:kag/gm

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

-25-

MSFT\17210AM3.DOC